# Можливості пакетів обробки різних типів даних в Python

**Пакет JSON**

Формат даних JSON (JavaScript Object Notation) може містити наступні дані всередині себе:

|  |  |
| --- | --- |
| JSON (JS) | Відповідність в Python |
| null | None |
| Number | int | float |
| String | str |
| Boolean (true | false) | True | False |
| Object | dict |
| Array | list |

Модуль json в Python відповідає за закодування і декодування даного типу представлення даних.

|  |
| --- |
| **import** json |

|  |  |
| --- | --- |
| **Функції пакету** | |
| json.dumps(obj) | Перетворює дані (словник obj), переданий функції в json формат. Функція повертає перетворений формат в вигляді рядка str.  Бере додаткові параметри, такі як:   * sort\_keys=bool – сортує ключі в об’єктах * indent=number – величина табуляції для читабельності файлу * separators – tuple з двома значеннями (item\_separator, key\_separator) (як розділяти елементи списку і об’єкту і як розділяти ключі об’єкта від їх значень) * default – функція, яку викликають, якщо в obj під час кодування натрапили на об’єкт, який не можна закодувати в json – не є одним з відповідних об’єктів в Python до типів json. |
| json.dump(obj, fp) | Функція має аналогічний функціонал до dumps, але не повертає нічого, а записує все в файл.  fp – об’єкт, типу попередньо відкритого файлу, який підтримує функцію .write() |
| json.loads(obj) | Бере стрічку obj, з закодованим форматом типу json і повертає представлення даних цього формату в стандартних типах Python.  Приймає наступні додаткові параметри:   * cls – особливий підклас JSONDecoder, для використання під час декодування інформації (default – JSONDecoder) * object\_hook – функція, яку буде викликано кожного разу, як json модуль декодує object в json (функція отримає словник, а повернене значення буде використане замість даного словника) * object\_pairs\_hook – функція з аналогічним функціоналом, тільки як параметр в неї буде передвавтись list з пар ключа – значення словника. * parse\_float, parse\_int, parse\_constant – функції будуть викликані при обробці заданих форматів даних. (До констат належать -Infinity, Infinity, NaN) |
| json.load(obj, fp) | Функція має аналогічний функціонал до loads, але бере всю інформацію з попередньо відкритого файлу (fp) |

В разі помилки під час декодування / енкодування інформації (зокрема, при неможливості перетворення між типами даних в Python, або невідомою частиною коду json), модуль видасть помилку **JSONDecodeError**.

**Пакет html.parser**

|  |
| --- |
| **from** html.parser **import** HTMLParser |

Більшість функціонала даного пакета заснована на класі HTMLParser, який зчитує формат html і водночас в залежності від зчитаних даних, викликає функції для їх обробки. Дані функції мають бути переписані користувачем, адже не має єдиного способу використання прочитаного html.

|  |  |
| --- | --- |
| **Методи класу HTMLParser, напиані для зчитування з html і готові для використання користувачем** | |
| feed(data) | Зчитує дані data, представлені в вигляді str, при цьому обробляючи інформацію |
| close() | Метод, який викликається по закінченню зчитування. Можна переписати для зміни функціоналу по закінченню зчитування, або використати для примусового завершення обробки інформації. |
| reset() | Зупиняє роботу – всі необроблені дані втрачаються. Викликається до початку обробки нової інформації, щоб обновити. |
| getpos() | Повертає позицію зчитування – рядок і букву рядка |
| get\_starttag\_text() | Повертає текст останнього відкритого тега |
| **Функції, які при успадкуванні класу варто переписати для визначення функціоналу класу** | |
| handle\_starttag(tag, attrs)  handle\_endtag(tag)  handle\_startendtag(tag, attrs) | Відповідають за обробку відкриваючих і закриваючих тегів  tag – назва тегу  attrs – словник, з значеннями атрибутів тегу |
| handle\_data(data) | Відповідає за обробку внутрішніх даних тегу (текст, код …) |
| handle\_comment(data) | Обробка коментарів (<!—comment -->) |
| Також: handle\_entityref(name), handle\_charref(name), handle\_decl(decl), handle\_pi(data), unknow\_decl(data) | |

**Пакет xml**

Пакет має багато функцій і способів для обробки xml. Один з них – за допомогою ElementTree.

|  |
| --- |
| **from** xml.etree **import** ElementTree as ET |

Цю частину пакету можна використовувати для зчитування xml з файлу або стрічки і потім – обробки отриманої інформації по кожному елементу. структура формату буде представлена деревом усіх його елементів, де кожен елемент містить інформацію про себе і посилання на його синів.

|  |  |
| --- | --- |
| **Функції пакета** | |
| tree = ET.parse(file\_name) | Зчитує xml з файлу за його локацією вказаною в аргументах функції  Для отримання кореня, необхідно використати .getroot() функцію |
| root = ET.fromstring(str) | Зчитує xml з str, повертає корінь дерева xml елементів |
| tree.write(‘output.xml’) | Записує дане дерево в файл. |
| **Функції і атрибути створеного об’єкта Element** | |
| tag | Назва тегу елемента |
| attrib | Словник атрибутів тегу елемента |
| text | Внутрішній вміст тегу (окрім інших тегів) |
| Сам елемент можна використовувати, як ітератор всіх його дочірніх елементів.   |  | | --- | | **for** child **in** element:  **print**(child.tag, child.attrib) | | |
| iter(tag\_name) | Використовується для ітерації по всіх нащадках даного елементу, тег який є такий самий, як і вказаний аргументом. |
| find(tag\_name) | Знаходить першого сина даного елемента, який має вказаний тег.  Функція findall(tag\_name) знаходить всіх синів з вказаним тегом |
| get(attr\_name) | Доступ до атрибутів даного елементу |
| set(attr\_name, value) | Змінює значення атрибута елемента |
| append(child) | Додає сина даному елементу |
| remove(child) | Забирає сина від батька |

Для одночасного зчитування і обробки інформації варто використовувати **ET.XMLPullParser([‘start’, ‘end’])**